AD-A214 900

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

Ada Compiler Validation Summary Report: SD-Scicon, plc, XD Ada MC68020 T1.0-02V, VAX Cluster (Host) to MC68020 (Target), 890321N1.10041

5. TYPE OF REPORT & PERIOD COVERED
21 March 1989 to 21 March 199

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

National Computing Centre Limited,
Manchester, United Kingdom.

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION AND ADDRESS

National Computing Centre Limited,
Manchester, United Kingdom.

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

Ada Joint Program Office
United States Department of Defense
Washington, DC 20301-3081

12. REPORT DATE

13. NUMBER OF PAGES

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

National Computing Centre Limited,
Manchester, United Kingdom.

15. SECURITY CLASS (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20 if different from Report)

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

SD-Scicon plc. XD Ada MC68020 TI.0-02V, Machester M1 7ED England, VAX Cluster comprising of a VAX 8600 and sen MicroVAX IIs under VMS 5.0 (Host) to MC68020 implemented on the MVME 133 XT board (bare machine)(Target), ACVC 1.10.

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

Ada Compiler Validation Summary Report:

Compiler Name: XD Ada MC68020 T1.0-02V

Certificate Number: #890321N1.10041

Host: VAX Cluster comprising of a VAX 8600 and seven MicroVAX IIs under VMS 5.0

Target: MC68020 implemented on the MVME 133 XT board (bare machine)

Testing Completed 21 March 1989 Using ACVC 1.10

This report has been reviewed and is approved.

J Pink

Jane Pink
Testing Services Manager
The National Computing Centre Limited
Oxford Road
Manchester M1 7ED
England

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Dr J. Solomon
Director AJPO
Department of Defense
Washington DC 20301

Ada Compiler Validation Summary Report:

Compiler Name: XD Ada MC68020 T1.0-02V

Certificate Number: #890321N1.10041

Host: VAX Cluster comprising of a VAX 8600 and seven MicroVAX IIs under VMS 5.0

Target: MC68020 implemented on the MVME 133 XT board (bare machine)

Testing Completed 21 March 1989 Using ACVC 1.10

This report has been reviewed and is approved.

Jane Pink
Testing Services Manager
The National Computing Centre Limited
Oxford Road
Manchester M1 7ED
England

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Dr J. Solomon
Director AJPO
Department of Defense
Washington DC 20301

AVF Control Number: AVF-VSR-90502/48

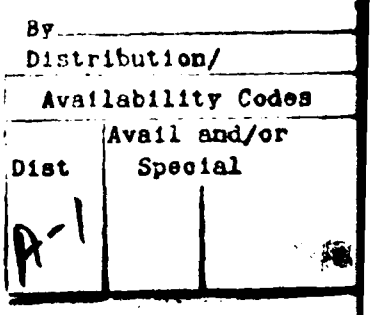
Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: #890321N1.10041
SD-Scicon plc
XD Ada MC68020 T1.0-02V
VAX Cluster Host and MC68020 Target

Completion of On-Site Testing:
21 March 1989

Prepared By:
Testing Services
The National Computing Centre Limited
Oxford Road
Manchester M1 7ED
England

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

TABLE OF CONTENTS



CHAPTER 1

INTRODUCTION

This Validation Summary Report ~~(VSR)~~ describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability ~~(ACVC)~~. An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies -- for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent, but is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by The National Computing Centre according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 21 March 1989 at SD-Scicon Plc, Pembroke House, Pembroke Broadway, Camberley, Surrey.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Stre
Washington DC 20301-3081

or from:

Testing Services
The National Computing Centre Limited
Oxford Road
Manchester M1 7ED
England

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.

4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada Commentary	An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures and Guidelines.

AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer which executes the code generated by the compiler.
Test	A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into

six test classes: A, B, C, D, E and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation and execution of legal Ada programs with certain language constructs which cannot be verified at run time. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check the run time system to ensure that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters -- for example, the number of identifiers permitted in a compilation or the number of units in a library -- a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a

Class F test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time -- that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated. In some cases, an implementation may legitimately detect errors during compilation of the test.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of each test in the ACVC follows conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values -- for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing

a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: XD Ada MC68020 T1.0-02V

ACVC Version: 1.10

Certificate Number: #890321N1.10041

Host Computer:

Machine:	VAX CLUSTER comprising of a VAX 8600 and seven MicroVAX II's.
Operating System:	VMS 5.0
Memory Size:	90Mb

Target Computer:

Machine:	MC68020 implemented on a MVME 133 XT board (bare machine)
Memory Size:	1Mb

Communications Network: RS232 link

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

a. Capacities.

(1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)

(2) The compiler correctly processes tests containing loop statements nested to 65 levels. (See tests D55A03A..H (8 tests).)

(3) The compiler correctly processes tests containing block statements nested to 65 levels. (See test D56001B.)

(4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to 17 levels. (See tests D64005E..G (3 tests))

b. Predefined types.

(1) This implementation supports the additional predefined types SHORT_INTEGER, SHORT_SHORT_INTEGER, LONG_FLOAT and LONG_LONG_FLOAT in the package STANDARD. (See tests B86001T..Z (7 tests).)

c. Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

(1) None of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)

(2) Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

(3) This implementation uses no extra bits for extra precision and uses all extra bits for extra range. (See test C35903A.)

(4) No exception is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

(5) No exception is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

(6) Underflow is gradual. (See tests C45524A..Z (26 tests).)

d. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

(1) The method used for rounding to integer is round to even. (See tests C46012A..Z (26 tests).)

(2) The method used for rounding to longest integer is round to even. (See tests C46012A..Z (26 tests).)

(3) The method used for rounding to integer in static universal real expressions is round away from zero. (See test C4A014A.)

e. Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT. For this implementation:

(1) Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises NUMERIC_ERROR. (See test C36003A.)

(2) NUMERIC_ERROR is raised when an array type with INTEGER'LAST + 2 components is declared. (See test C36202A.)

(3) NUMERIC_ERROR is raised when an array type with SYSTEM.MAX_INT + 2 components is declared. (See test C36202B.)

(4) A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when the array type is declared. (See test C52103X.)

(5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array type is declared. (See test C52104Y.)

(6) In assigning one-dimensional array types, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

(7) In assigning two-dimensional array types, the expression is not evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

f. A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

g. Discriminated types.

(1) In assigning record types with discriminants, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

h. Aggregates.

(1) In the evaluation of a multi-dimensional aggregate, the test results indicate that all choices are evaluated before checking against the index type. (See tests C43207A and C43207B.)

(2) In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

(3) CONSTRAINT_ERROR is raised after all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

i. Pragmas.

(1) The pragma INLINE is supported for functions or procedures. (See tests LA3004A..B (2 tests), EA3004C..D (2 tests), and CA3004E..F (2 tests).)

j. Generics.

(1) Generic specifications and bodies can be compiled in separate compilations. (See tests CA1012A, CA2009C, CA2009F, BC3204C, and BC3205D.)

(2) Generic subprogram declarations and bodies can be compiled in separate compilations. (See tests CA1012A and CA2009F.)

(3) Generic library subprogram specifications and bodies can be compiled in separate compilations. (See test CA1012A.)

(4) Generic non-library package bodies as subunits can be compiled in separate compilations. (See test CA2009C.)

(5) Generic non-library subprogram bodies can be compiled in separate compilations from their stubs. (See test CA2009F.)

(6) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

(7) Generic package declarations and bodies can be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)

(8) Generic library package specifications and bodies can be compiled in separate compilations. (See tests BC3204C and BC3205D.)

(9) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

k. Input and output.

(1) The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types or record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

(2) The package DIRECT_IO cannot be instantiated with unconstrained array types or record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

(3) The director, AJPO, has determined (AI-00332) that every call to OPEN and CREATE must raise USE_ERROR or NAME_ERROR if file input/output is not supported. This implementation exhibits this behavior for SEQUENTIAL_IO, DIRECT_IO, and TEXT_IO.

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 43 tests had been withdrawn because of test errors. The AVF determined that 550 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 159 executable tests that use floating-point precision exceeding that supported by the implementation. Modifications to the code, processing, or grading for 12 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	126	1133	1787	17	14	46	3124
Inapplicable	3	5	528	0	14	0	550
Withdrawn	1	2	34	0	6	0	43
TOTAL	130	1140	2350	17	34	46	3717

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Pass	201	591	568	245	172	99	161	332	137	36	252	254	76	3124
N/A	11	58	112	3	0	0	5	1	0	0	0	115	245	550
W/D	1	1	0	0	0	0	0	1	0	0	1	35	4	43
TOT	213	650	680	248	172	99	166	334	137	36	253	404	325	3717

3.4 WITHDRAWN TESTS

The following 43 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

E28005C	A39005G
B97102E	BC3009B
CD2A62D	CD2A63A..D (4 tests)
CD2A66A..D (4 tests)	CD2A73A..D (4 tests)
CD2A76A..D (4 tests)	CD2A81G
CD2A83G	CD2A84N..M (2 tests)
CD5011O	CD2B15C
CD7205C	CD2D11B
CD5007B	ED7004B
ED7005C..D (2 tests)	ED7006C..D (2 tests)
CD7105A	CD7203B
CD7204B	CD7205D
CE2107I	CE3111C
CE3301A	CE3411B

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 550 tests were inapplicable for the reasons indicated:

a. The following 159 tests are not applicable because they have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113O..Y (11 tests)	C35705O..Y (11 tests)
C35706O..Y (11 tests)	C35707O..Y (11 tests)
C35708O..Y (11 tests)	C35802O..Z (12 tests)
C45241O..Y (11 tests)	C45321O..Y (11 tests)
C45421O..Y (11 tests)	C45521O..Z (12 tests)
C45524O..Z (12 tests)	C45621O..Z (12 tests)
C45641O..Y (11 tests)	C46012O..Z (12 tests)

b. The following 16 tests are not applicable because 'SMALL representation clauses are not supported.

A39005E	C87B62C
CD1009L	CD1C03F
CD2A53A..E (5 tests)	CD2A54A..B (2 tests)
CD2A54G	CD2A54I
ED2A56A	CD2D11A
CD2D13A	

c. C35702A and B86001T are not applicable because this implementation supports no predefined type SHORT_FLOAT.

d. The following 16 tests are not applicable because this implementation does not support a predefined type LONG_INTEGER:

C45231C	C45304C	C45502C	C45503C	C45504C
C45504F	C45611C	C45613C	C45614C	C45631C
C45632C	B52004D	C55B07A	B55B09C	B86001W
CD7101F				

e. C45531M..P (4 tests) and C45532M..P (4 tests) are all inapplicable because this implementation has a 'MAX_MANTISSA of 31 and these tests require the compiler to support a greater value.

f. C4A013B is not applicable because the evaluation of an expression involving 'MACHINE_RADIX applied to the most precise floating-point type would raise an exception; since the expression must be static, it is rejected at compile time.

g. B86001Y is not applicable because this implementation supports no predefined fixed-point type other than DURATION.

h. C86001F is not applicable because package system is used by TEXT_IO.

i. C96005B is not applicable because there are no values of type DURATION'BASE that are outside the range of DURATION.

j. CD1009C, CD2A41A..E (5 tests) and CD2A42A..J (10 tests) are not applicable because 'SIZE representation clauses for floating-point types are not supported.

k. The following 25 tests are not applicable because 'SIZE representation clauses for array types are not supported.

CD1009E..F (2 tests) CD2A61A..L (12 tests)
CD2A62A..C (3 tests) CD2A64A..D (4 tests)
CD2A65A..D (4 tests)

l. The following 17 tests are not applicable because 'SIZE representation clauses for record types are not supported.

CD1009G CD2A71A..D (4 tests)
CD2A72A..D (4 tests) CD2A74A..D (4 tests)
CD2A75A..D (4 tests)

m. CD1C04C is inapplicable for because this implementation does not support model numbers of a derived type that are not representable values of the parent type.

n. CD2A52J and CD2A54J are not applicable because these tests require an unsigned representation for a fixed point type; this implementation does not support unsigned fixed point representation.

o. CD2A52C..D (2 tests), CD2A52G..H (2 tests), CDA54C..D (2 tests) and CD2A54H are not applicable because for this implementation the legality of a 'SIZE clause for a derived fixed point type can depend on the representation chosen for the parent type.

p. The following 23 tests are not applicable because 'SIZE representation clauses for access types are not supported.

CD2A81A..F (6 tests) CD2A83A..C (3 tests)
CD2A83E..F (2 tests) CD2A84B..I (8 tests)
CD2A84K..L (2 tests) ED2A86A
CD2A87A

q. CD2A91A..E (5 tests) are not applicable because 'SIZE representation clauses for task types are not supported.

r. CD2B15B is not applicable because this implementation allows 'STORAGE_SIZE to yield the size requested by the user

s. CD7004C, CD7005E and CD7006E are all not applicable because of other limitations, not specified in the Referance Manual, on the use of the pragmas SYSTEM_NAME, STORAGE_UNIT and MEMORY_SIZE.

t. AE2101C, EE2201D, and EE2201E use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types with discriminants without defaults. These instantiations are rejected by this compiler.

u. AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types with discriminants without defaults. These instantiations are rejected by this compiler.

v. The following 236 tests are inapplicable because sequential, text, and direct access files are not supported:

CE2102A..C (3 tests)	CE2102G..H (2 tests)
CE2102K	CE2102N..Y (12 tests)
CE2103C..D (2 tests)	CE2104A..D (4 tests)
CE2105A..B (2 tests)	CE2106A..B (2 tests)
CE2107A..H (8 tests)	CE2107L
CE2108A..B (2 tests)	CE2108C..H (6 tests)
CE2109A..C (3 tests)	CE2110A..D (4 tests)
CE2111A..I (9 tests)	CE2115A..B (2 tests)
CE2201A..C (3 tests)	CE2201F..N (9 tests)
CE2204A..D (4 tests)	CE2205A
CE2208B	CE2401A..C (3 tests)
CE2401E..F (2 tests)	CE2401H..L (5 tests)
CE2404A..B (2 tests)	CE2405B
CE2406A	CE2407A..B (2 tests)
CE2408A..B (2 tests)	CE2409A..B (2 tests)
CE2410A..B (2 tests)	CE2411A
CE3102A..B (2 tests)	EE3102C
CE3102F..H (3 tests)	CE3102J..K (2 tests)
CE3103A	CE3104A..C (3 tests)
CE3107B	CE3108A..B (2 tests)
CE3109A	CE3110A
CE3111A..B (2 tests)	CE3111D..E (2 tests)
CE3112A..D (4 tests)	CE3114A..B (2 tests)
CE3115A	EE3203A
CE3208A	EE3301B
CE3302A	CE3305A
CE3402A	EE3402B
CE3402C..D (2 tests)	CE3403A..C (3 tests)
CE3403E..F (2 tests)	CE3404B..D (3 tests)
CE3405A	EE3405B
CE3405C..D (2 tests)	CE3406A..D (4 tests)
CE3407A..C (3 tests)	CE3408A..C (3 tests)
CE3409A	CE3409C..E (3 tests)
EE3409F	CE3410A
CE3410C..E (3 tests)	EE3410F
CE3411A,C (2 tests)	CE3412A
CE3413A	CE3413C
CE3602A..D (4 tests)	CE3603A
CE3604A..B (2 tests)	CE3605A..E (5 tests)
CE3606A..B (2 tests)	CE3704A..F (6 tests)

CE3704M..O (3 tests) CE3706D
CE3706F..G (2 tests) CE3804A..P (16 tests)
CE3805A..B (2 tests) CE3806A..B (2 tests)
CE3806D..E (2 tests) CE3806H
CE3905A..C (3 tests) CE3905L
CE3906A..C (3 tests) CE3906E..F (2 tests)

w. CE3806G and CE3901A are inapplicable because this implementation raises NAME_ERROR on the attempt to create a text file with a non-null filename.

x. EE3412C is not applicable because this implementation's body of the package REPORT does not use TEXT_IO.

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that was not anticipated by the test (such as raising one exception instead of another).

Modifications were required for 15 tests.

C45524A..N (14 tests) were modified because these tests expect that the result of continued division of a real number will be zero; the Ada Standard, however, only requires that the result be within the type's SAFE_SMALL of zero. Thus, these tests were modified to include a check that the result was in the smallest positive safe interval for the type. The implementation passed the modified tests. Each test was modified by inserting the following code after line 138;

```
ELSIF VAL <= F'SAFE_SMALL THEN
     COMMENT ("UNDERFLOW IS GRADUAL")
```

The following test was split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B97103E

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the VAX/VMS x MC68020 XD Ada compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the VAX/VMS x MC68020 XD Ada compiler using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware and software components:

Host computer	: VAX Cluster comprising of a VAX 8600 and seven Micro VAX IIs
Host operating system	: VMS 5.0
Target computer	: MC68020 implemented on a MVME 133 XT board (bare machine)
Compiler	: XD Ada MC68020 T1.0-02V
Assembler	: XD Ada MC68020 T1.0-02V
Linker	: XD Ada MC68020 T1.0-02V
Loader/Downloader	: XD Ada MC68020 T1.0-02V
Runtime System	: XD Ada MC68020 T1.0-02V

The host and target computers were linked via a RS232 connector.

A magnetic tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were not included in their modified form on the magnetic tape.

The contents of the magnetic tape were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled and linked on the VAX Cluster, then all executable images were transferred to the MC68020 via the RS232 link and run. Results were printed from the host computer.

The compiler was tested using command scripts provided by SD-Scicon and reviewed by the validation team. The compiler was tested using all following option settings:-

OPTION	EFFECT
DEBUG=ALL	Both debugger symbol records and traceback information are included in the object file
OPT=(TIME,INLINE:NORMAL)	Provides full optimisation with time as primary objective and also provides normal subprogram expansion
CHECK	Overrides all the suppressing programs in the source code
NOANALYSIS_CHECK	No cross reference file is created
COPY_SOURCE	Creates a file containing a copy of the source code when errors are found
NODIAGNOSTICS	Does not create a file to contain the diagnostic messages from the computer
LIST	A Listing file is created
NOMACHINE_CODE	No machine code to be included in he Listing file
NOTE_SOURCE	Note's the file specification or the source code in the program library
SHOW = PORTABILITY	Includes a program portability summary in the Listing file
NOSYNTAX_ONLY	The compiler performs all checks on the source code

Tests were compiled, linked, and executed (as appropriate) using a cluster of 8 computers and 2 target computers. Test output, compilation listings, and job logs were captured on magnetic

media and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at SD-Scicon plc, Pembroke House, Pembroke Broadway, Camberley, Surrey and was completed on 21 March 1989.

APPENDIX A

DECLARATION OF CONFORMANCE

SD-Scicon plc has submitted the following Declaration of Conformance concerning the XD Ada MC68020 T1.0-02V compiler.

DECLARATION OF CONFORMANCE

Compiler Implementor: SD-Scicon plc

Ada Validation Facility: The National Computing Centre
Oxford Road
Manchester M1 7ED
England

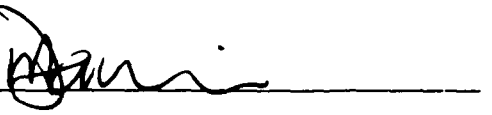
Ada Compiler Validation Capability (ACVC) Version: 1.10

Base Configuration

Base Compiler Name:	XD Ada MC68020 T1.0-02V
Host Architecture:	VAX CLUSTER comprising of a VAX 8600 and seven MicroVAX II's
Host OS and Version:	VMS 5.0
Target Architecture:	MC68020 inplemented on the MVME 133 XT board (bare machine)

Implementor's Declaration

I, the undersigned, representing SD-Scicon plc, have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler(s) listed in this declaration. I declare that SD-Scicon plc is the owner of record of the Ada language compiler(s) listed above and, as such, is responsible for maintaining said compiler(s) in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the owner's corporate name.



Date : 21/3/89

Owner's Declaration

I, the undersigned, representing SD-Scicon plc, take full responsibility for implementation and maintenance of the Ada compiler(s) listed above, and agree to the public disclosure of the final Validation Summary Report. I declare that all of the Ada language compilers listed, and their host/target performance, are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.

W [illegible] Date : 21/3/89

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the **XD Ada MC68020 T1.0-02V** compiler, as described in this Appendix, are provided by **SD-Scicon plc**. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is

...

type INTEGER is range -2147483648 .. 2147483647;
type SHORT_INTEGER is range -32768 .. 32767;
type SHORT_SHORT_INTEGER is range -128 .. 127;

type FLOAT is digits 6 range -(2**128 - 2**104) ..
                                2**128 - 2**104;
type LONG_FLOAT is digits 15 range -(2**1024 - 2**971) ..
                                    2**1024 - 2**971;

type  DURATION  is  delta  1.E-4  range  -131072.0000  ..
                                         131071.9999;

...

end STANDARD;
```

Appendix F

Implementation-Dependent Characteristics

NOTE

This appendix is not part of the standard definition of the Ada programming language.

This appendix summarizes the following implementation-dependent characteristics of XD Ada:

- Listing the XD Ada pragmas and attributes.
- Giving the specification of the package SYSTEM.
- Presenting the restrictions on representation clauses and unchecked type conversions.
- Giving the conventions for names denoting implementation-dependent components in record representation clauses.
- Giving the interpretation of expressions in address clauses.
- Presenting the implementation-dependent characteristics of the input-output packages.
- Presenting other implementation-dependent characteristics.

F.1 Implementation-Dependent Pragmas

XD Ada provides the following pragmas, which are defined elsewhere in the text. In addition, XD Ada restricts the predefined language pragmas INLINE and INTERFACE, replaces pragma SHARED with pragma VOLATILE, and provides pragma SUPPRESS_ALL in addition to pragma SUPPRESS. See Appendix B for a descriptive pragma summary.

- EXPORT_EXCEPTION (see Section 13.9a.3.2)
- EXPORT_FUNCTION (see Section 13.9a.1.2)
- EXPORT_OBJECT (see Section 13.9a.2.2)
- EXPORT_PROCEDURE (see Section 13.9a.1.2)
- IMPORT_EXCEPTION (see Section 13.9a.3.1)
- IMPORT_FUNCTION (see Section 13.9a.1.1)
- IMPORT_OBJECT (see Section 13.9a.2.1)
- IMPORT_PROCEDURE (see Section 13.9a.1.1)
- LEVEL (see Section 13.5.1)
- LINK_OPTION (see Appendix B)
- MACHINE_CODE_PROCEDURE (see Section 13.8)
- SUPPRESS_ALL (see Section 11.7)
- TITLE (see Annex B)
- VOLATILE (see Section 9.11)

F.2 Implementation-Dependent Attributes

XD Ada provides the following attributes, which are defined elsewhere in the text. See Appendix A for a descriptive attribute summary.

- BIT (see Section 13.7.2)
- MACHINE_SIZE (see Section 13.7.2)
- TYPE_CLASS (see Section 13.7a.2)

F.3 Specification of the Package System

The package SYSTEM for the MC68020 is as follows:

F.3.1 Package System for MC68020 Target

```
package SYSTEM is
    type NAME is (MC68020, MIL_STD_1750A);

    SYSTEM_NAME    : constant NAME := MC68020;
    STORAGE_UNIT   : constant := 8;
    MEMORY_SIZE    : constant := 2**32;
    MIN_INT        : constant := -(2**31);
    MAX_INT        : constant := 2**31-1;
    MAX_DIGITS     : constant := 18;
    MAX_MANTISSA   : constant := 31;
    FINE_DELTA     : constant := 2.0**(-31);
    TICK           : constant := 162.5E-6
    subtype PRIORITY is INTEGER range 0 .. 15;

    subtype LEVEL is INTEGER range 0 .. 7;

    type ADDRESS_INT is range MIN_INT .. MAX_INT;

--  Address type

    type ADDRESS is private;

    ADDRESS_ZERO : constant ADDRESS;
--

    function TO_ADDRESS (ADDR : universal_integer) return ADDRESS;

--

    function CONVERT_ADDRESS (ADDR : ADDRESS) return ADDRESS_INT;
    function CONVERT_ADDRESS (ADDR : ADDRESS_INT) return ADDRESS;

    function "+" (LEFT : ADDRESS; RIGHT : ADDRESS_INT) return ADDRESS;
    function "+" (LEFT : ADDRESS_INT; RIGHT : ADDRESS) return ADDRESS;
    function "-" (LEFT : ADDRESS; RIGHT : ADDRESS) return ADDRESS_INT;
    function "-" (LEFT : ADDRESS; RIGHT : ADDRESS_INT) return ADDRESS;

--  function "="  (LEFT, RIGHT : ADDRESS) return BOOLEAN;
--  function "/=" (LEFT, RIGHT : ADDRESS) return BOOLEAN;
    function "<"  (LEFT, RIGHT : ADDRESS) return BOOLEAN;
    function "<=" (LEFT, RIGHT : ADDRESS) return BOOLEAN;
    function ">"  (LEFT, RIGHT : ADDRESS) return BOOLEAN;
    function ">=" (LEFT, RIGHT : ADDRESS) return BOOLEAN;

--  Note that because ADDRESS is a private type
--  the functions "=" and "/=" are already available

-- Generic functions used to access memory
```

```
    generic
        type TARGET is private;
    function FETCH_FROM_ADDRESS (A : ADDRESS) return TARGET;

    generic
        type TARGET is private;
    procedure ASSIGN_TO_ADDRESS (A : ADDRESS; T : TARGET);

    type TYPE_CLASS is (TYPE_CLASS_ENUMERATION,
                        TYPE_CLASS_INTEGER,
                        TYPE_CLASS_FIXED_POINT,
                        TYPE_CLASS_FLOATING_POINT,
                        TYPE_CLASS_ARRAY,
                        TYPE_CLASS_RECORD,
                        TYPE_CLASS_ACCESS,
                        TYPE_CLASS_TASK,
                        TYPE_CLASS_ADDRESS);

--  XD Ada hardware-oriented types and functions
    type    BIT_ARRAY is array (INTEGER range <>) of BOOLEAN;
    pragma  PACK(BIT_ARRAY);
    subtype BIT_ARRAY_8 is BIT_ARRAY  (0 ..  7);
    subtype BIT_ARRAY_16 is BIT_ARRAY (0 .. 15);
    subtype BIT_ARRAY_32 is BIT_ARRAY (0 .. 31);
    subtype BIT_ARRAY_64 is BIT_ARRAY (0 .. 63);
    type UNSIGNED_BYTE  is range 0 .. 255;
    for  UNSIGNED_BYTE'SIZE  use 8;
    function "not" (LEFT       : UNSIGNED_BYTE) return UNSIGNED_BYTE;
    function "and" (LEFT, RIGHT : UNSIGNED_BYTE) return UNSIGNED_BYTE;
    function "or"  (LEFT, RIGHT : UNSIGNED_BYTE) return UNSIGNED_BYTE;
    function "xor" (LEFT, RIGHT : UNSIGNED_BYTE) return UNSIGNED_BYTE;

    function TO_UNSIGNED_BYTE (LEFT : BIT_ARRAY_8) return UNSIGNED_BYTE;
    function TO_BIT_ARRAY_8 (LEFT : UNSIGNED_BYTE) return BIT_ARRAY_8;
    type UNSIGNED_BYTE_ARRAY is array (INTEGER range <>) of UNSIGNED_BYTE;
    type UNSIGNED_WORD  is range 0 .. 65535;
    for  UNSIGNED_WORD'SIZE  use 16;
    function "not" (LEFT       : UNSIGNED_WORD) return UNSIGNED_WORD;
    function "and" (LEFT, RIGHT : UNSIGNED_WORD) return UNSIGNED_WORD;
    function "or"  (LEFT, RIGHT : UNSIGNED_WORD) return UNSIGNED_WORD;
    function "xor" (LEFT, RIGHT : UNSIGNED_WORD) return UNSIGNED_WORD;

    function TO_UNSIGNED_WORD (LEFT : BIT_ARRAY_16) return UNSIGNED_WORD;
    function TO_BIT_ARRAY_16 (LEFT : UNSIGNED_WORD) return BIT_ARRAY_16;
    type UNSIGNED_WORD_ARRAY is array (INTEGER range <>) of UNSIGNED_WORD;

--
    type UNSIGNED_LONGWORD is range MIN_INT .. MAX_INT;

    function "not" (LEFT       : UNSIGNED_LONGWORD) return UNSIGNED_LONGWORD;
    function "and" (LEFT, RIGHT : UNSIGNED_LONGWORD) return UNSIGNED_LONGWORD;
    function "or"  (LEFT, RIGHT : UNSIGNED_LONGWORD) return UNSIGNED_LONGWORD;
    function "xor" (LEFT, RIGHT : UNSIGNED_LONGWORD) return UNSIGNED_LONGWORD;
```

```
    function TO_UNSIGNED_LONGWORD (LEFT : BIT_ARRAY_32) return UNSIGNED_LONGWORD;
    function TO_BIT_ARRAY_32 (LEFT : UNSIGNED_WORD) return BIT_ARRAY_32;
    type UNSIGNED_LONGWORD_ARRAY is array (INTEGER range <>) of UNSIGNED_LONGWORD;

--  Conventional names for static subtypes of type UNSIGNED_LONGWORD
    subtype UNSIGNED_1  is UNSIGNED_LONGWORD range 0 .. 2** 1-1;
    subtype UNSIGNED_2  is UNSIGNED_LONGWORD range 0 .. 2** 2-1;
    subtype UNSIGNED_3  is UNSIGNED_LONGWORD range 0 .. 2** 3-1;
    subtype UNSIGNED_4  is UNSIGNED_LONGWORD range 0 .. 2** 4-1;
    subtype UNSIGNED_5  is UNSIGNED_LONGWORD range 0 .. 2** 5-1;
    subtype UNSIGNED_6  is UNSIGNED_LONGWORD range 0 .. 2** 6-1;
    subtype UNSIGNED_7  is UNSIGNED_LONGWORD range 0 .. 2** 7-1;
    subtype UNSIGNED_8  is UNSIGNED_LONGWORD range 0 .. 2** 8-1;
    subtype UNSIGNED_9  is UNSIGNED_LONGWORD range 0 .. 2** 9-1;
    subtype UNSIGNED_10 is UNSIGNED_LONGWORD range 0 .. 2**10-1;
    subtype UNSIGNED_11 is UNSIGNED_LONGWORD range 0 .. 2**11-1;
    subtype UNSIGNED_12 is UNSIGNED_LONGWORD range 0 .. 2**12-1;
    subtype UNSIGNED_13 is UNSIGNED_LONGWORD range 0 .. 2**13-1;
    subtype UNSIGNED_14 is UNSIGNED_LONGWORD range 0 .. 2**14-1;
    subtype UNSIGNED_15 is UNSIGNED_LONGWORD range 0 .. 2**15-1;
    subtype UNSIGNED_16 is UNSIGNED_LONGWORD range 0 .. 2**16-1;
    subtype UNSIGNED_17 is UNSIGNED_LONGWORD range 0 .. 2**17-1;
    subtype UNSIGNED_18 is UNSIGNED_LONGWORD range 0 .. 2**18-1;
    subtype UNSIGNED_19 is UNSIGNED_LONGWORD range 0 .. 2**19-1;
    subtype UNSIGNED_20 is UNSIGNED_LONGWORD range 0 .. 2**20-1;
    subtype UNSIGNED_21 is UNSIGNED_LONGWORD range 0 .. 2**21-1;
    subtype UNSIGNED_22 is UNSIGNED_LONGWORD range 0 .. 2**22-1;
    subtype UNSIGNED_23 is UNSIGNED_LONGWORD range 0 .. 2**23-1;
    subtype UNSIGNED_24 is UNSIGNED_LONGWORD range 0 .. 2**24-1;
    subtype UNSIGNED_25 is UNSIGNED_LONGWORD range 0 .. 2**25-1;
    subtype UNSIGNED_26 is UNSIGNED_LONGWORD range 0 .. 2**26-1;
    subtype UNSIGNED_27 is UNSIGNED_LONGWORD range 0 .. 2**27-1;
    subtype UNSIGNED_28 is UNSIGNED_LONGWORD range 0 .. 2**28-1;
    subtype UNSIGNED_29 is UNSIGNED_LONGWORD range 0 .. 2**29-1;
    subtype UNSIGNED_30 is UNSIGNED_LONGWORD range 0 .. 2**30-1;
    subtype UNSIGNED_31 is UNSIGNED_LONGWORD range 0 .. 2**31-1;

private
    -- Not shown
end SYSTEM;
```

F.4 Restrictions on Representation Clauses

The representation clauses allowed in XD Ada are length, enumeration, record representation, and address clauses.

In XD Ada, a representation clause for a generic formal type or a type that depends on a generic formal type is not allowed. In addition, a representation clause for a composite type that has a component or subcomponent of a generic formal type or a type derived from a generic formal type is not allowed.

Restrictions on length clauses are specified in 13.2; restrictions on enumeration representation clauses are specified in Section 13.3; and restrictions on record representation clauses are specified in Section 13.4.

F.5 Conventions for Implementation-Generated Names Denoting Implementation-Dependent Components in Record Representation Clauses

XD Ada does not allocate implementation-dependent components in records.

F.6 Interpretation of Expressions Appearing in Address Clauses

Expressions appearing in address clauses must be of the type ADDRESS defined in package SYSTEM (see Section 13.7a.1 and Section F.3).

XD Ada allows address clauses for variables (see Section 13.5). For address clauses on variables, the address expression is interpreted as a Motorola full 32-bit address.

XD Ada supports address clauses on task entries to allow interrupts to cause a reschedule directly. For address clauses on task entries, the address expression is interpreted as a Motorola exception vector offset.

In XD Ada for the MC68020 family, values of type SYSTEM.ADDRESS are interpreted as integers in the range 0 .. $2^{32}-1$. As SYSTEM.ADDRESS is a private type, the only operations allowed on objects of this type are those given in package SYSTEM.

F.7 Restrictions on Unchecked Type Conversions

XD Ada supports the generic function UNCHECKED_CONVERSION with the restrictions given in Section 13.10.2.

F.8 Implementation-Dependent Characteristics of Input-Output Packages

The packages SEQUENTIAL_IO and DIRECT_IO are implemented as null packages that conform to the specification given in the LRM. The packages raise the exceptions specified in Chapter 14 of the LRM. The two possible exceptions that are raised by these packages are given here, in the order in which they are raised.

Exception	When Raised
STATUS_ERROR	Raised by an attempt to operate upon a file that is not open (no files can be opened).
USE_ERROR	Raised if exception STATUS_ERROR is not raised.

MODE_ERROR cannot be raised since no file can be opened (therefore it cannot have a current mode) and NAME_ERROR cannot be raised since there are no restrictions on file names.

The predefined package LOW_LEVEL_IO is not provided.

F.8.1 The Package TEXT_IO

The package TEXT_IO conforms to the specification given in the LRM. String input-output is implemented as defined. File input-output is supported to STANDARD_INPUT and STANDARD_OUTPUT only. The possible exceptions that are raised by package TEXT_IO are as follows:

Exception	When Raised
STATUS_ERROR	Raised by an attempt to operate upon a file that is not open (no files can be opened).
MODE_ERROR	Raised by an attempt to read from, or test for the end of, STANDARD_OUTPUT, or to write to STANDARD_INPUT
END_ERROR	Raised by an attempt to read past the end of STANDARD_INPUT.
USE_ERROR	Raised when an unsupported operation is attempted, that would otherwise be legal.

NAME_ERROR cannot be raised since there are no restrictions on file names.

The type COUNT is defined as follows:

```
type COUNT is range 0 .. INTEGER'LAST;
```

The subtype FIELD is defined as follows:

```
type FIELD is INTEGER range 0 .. 132;
```

F.8.2 The Package IO_EXCEPTIONS

The specification of the package IO_EXCEPTIONS is the same as that given in the LRM.

F.9 Other Implementation Characteristics

Implementation characteristics associated with the definition of a main program, various numeric ranges, and implementation limits are summarized in the following sections.

F.9.1 Definition of a Main Program

Any library procedure can be used as a main program provided that it has no formal parameters.

F.9.2 Values of Integer Attributes

The ranges of values for integer types declared in package STANDARD are as follows:

SHORT_SHORT_INTEGER	$-2^{7} .. 2^{7}-1$	(-128 .. 127)
SHORT_INTEGER	$-2^{15} .. 2^{15}-1$	(-32768 .. 32767)
INTEGER	$-2^{31} .. 2^{31}-1$	(-2147483648 .. 2147483647)

For the package TEXT_IO, the range of values for types COUNT and FIELD are as follows:

COUNT	$0 .. 2^{31}-1$	(0 .. 2147483647)
FIELD	0 .. 132	

F.9.3 Values of Floating-Point Attributes

Floating-point types are described in Section 3.5.7. The representation attributes of floating-point types are summarized in the following table:

	FLOAT	LONG_FLOAT	LONG_LONG_FLOAT
DIGITS	6	15	18
SIZE	32	64	96
MANTISSA	21	51	61
EMAX	84	204	244
EPSILON	2^{-20}	2^{-50}	2^{-60}
SMALL	2^{-85}	2^{-205}	2^{-245}
LARGE	$2^{84}-2^{63}$	$2^{204}-2^{153}$	$2^{244}-2^{183}$
SAFE_EMAX	125	1021	16382
SAFE_SMALL	2^{-126}	2^{-1022}	2^{-16383}
SAFE_LARGE	$2^{125}-2^{104}$	$2^{1021}-2^{970}$	$2^{16382}-2^{16321}$
FIRST	$-(2^{128}-2^{104})$	$-(2^{1024}-2^{971})$	$-(2^{16384}-2^{16320})$
LAST	$2^{128}-2^{104}$	$2^{1024}-2^{971}$	$2^{16384}-2^{16320}$
MACHINE_RADIX	2	2	2
MACHINE_MANTISSA	24	53	64
MACHINE_EMAX	128	1024	16384
MACHINE_EMIN	-125	-1021	-16382
MACHINE_ROUNDS	FALSE	FALSE	FALSE
MACHINE_OVERFLOWS	FALSE	FALSE	FALSE

F.9.4 Attributes of Type DURATION

The values of the significant attributes of type DURATION are as follows:

DURATION'DELTA	1.E-4	(10^{-4})
DURATION'SMALL	2#1.0#E-14	(2^{-14})
DURATION'FIRST	-131072.0000	(-2^{17})
DURATION'LAST	131071.9999	(2^{17} - DURATION'DELTA)

F.9.5 Implementation Limits

Limit	Description
120	Maximum identifier length (number of characters)
120	Maximum number of characters in a source line
2^{10}	Maximum number of library units and subunits in a compilation closure[1]
2^{12}	Maximum number of library units and subunits in an execution closure[2]
2^{16} -1	Maximum number of enumeration literals in an enumeration type definition
2^{16} -1	Maximum number of lines in a source file
2^{31} -1	Maximum number of bits in any object
2^{16} -1	Maximum number of exceptions

[1]The compilation closure of a given unit is the total set of units that the given unit depends on, directly and indirectly.

[2]The execution closure of a given unit is the compilation closure plus all associated secondary units.

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below:

Name and Meaning	Value
$ACC_SIZE An integer literal whose value is the number of bits sufficient to hold any value of an access type.	32
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	(1..254=>'A', 255=>1)
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	(1..254=>'A', 255=>2)
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	(1..127=>'A', 128=>3, 129..255=>'A')
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	(1..127=>'A', 128=>4, 129..255=>'A')
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length	(1..252=>0, 253..255=>298)

$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	(1..249=>0, 250..255=>69.0E1)
$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	(1..127=>'A')
$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	(1..127=>'A', 128=>1)
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	(1..235=>' ')
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2147483647
$DEFAULT_MEM_SIZE An integer literal whose value is SYSTEM.MEMORY_SIZE.	2147483647
$DEFAULT_STOR_UNIT An integer literal whose value is SYSTEM.STORAGE_UNIT.	8
$DEFAULT_SYS_NAME The value of the constant SYSTEM.SYSTEM_NAME.	MC68020
$DELTA_DOC A real literal whose value is SYSTEM.FINE_DELTA.	2#1.0#E-31
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	255
$FIXED_NAME The name of a predefined fixed-point type other than DURATION.	NO_SUCH_TYPE

$FLOAT_NAME The name of a predefined floating-point type other than FLOAT, SHORT_FLOAT, or LONG_FLOAT.	LONG_LONG_FLOAT
$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	131072.0
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	131073.0
$HIGH_PRIORITY An integer literal whose value is the upper bound of the range for the subtype SYSTEM.PRIORITY.	15
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	THERE ARE NO ILLEGAL FILENAMES
$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	N/A
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-2147483648
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	2147483647
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST+1.	2147483648
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-131072.0
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-131073.0

$LOW_PRIORITY An integer literal whose value is the lower bound of the range for the subtype SYSTEM.PRIORITY.	0
$MANTISSA_DOC An integer literal whose value is SYSTEM.MAX_MANTISSA.	31
$MAX_DIGITS Maximum digits supported for floating-point types.	18
$MAX_IN_LEN Maximum input line length permitted by the implementation.	255
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2147483647
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2147483648
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	(1..2=>'2:', 3..252=>'0', 253..255=>'11:')
$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	(1..3=>'16:' 4..251=>'0', 252..255=>'F.E:')
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	(1=>'"', 2..254=>'A', 255=>'"')
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-2147483648

$MIN_TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has no entries, no declarations, and "NULL;" as the only statement in its body.	32
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	SHORT_SHORT_INTEGER
$NAME_LIST A list of enumeration literals in the type SYSTEM.NAME, separated by commas.	MC68020
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFF_FFFF#
$NEW_MEM_SIZE An integer literal whose value is a permitted argument for pragma memory_size, other than $DEFAULT_MEM_SIZE. If there is no other value, then use $DEFAULT_MEM_SIZE.	123456
$NEW_STOR_UNIT An integer literal whose value is a permitted argument for pragma storage_unit, other than $DEFAULT_STOR_UNIT. If there is no other permitted value, then use value of SYSTEM.STORAGE_UNIT.	8
$NEW_SYS_NAME A value of the type SYSTEM.NAME, other than $DEFAULT_SYS_NAME. If there is only one value of that type, then use that value.	MC68020

$TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has a single entry with one inout parameter.	32
$TICK A real literal whose value is SYSTEM.TICK.	162.5E-6

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 43 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form AI-ddddd is to an Ada Commentary.

E28005C This test expects that the string "-- TOP OF PAGE. --63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this that must appear at the top of the page.

A39005G This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).

B97102E This test contains an unitended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).

BC3009B This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).

CD2A62D This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).

CD2A63A..D, CD2A66A..D, CD2A73A..D, CD2A76A..D [16 tests]
These tests wrongly attempt to check the size of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD2A81G, CD2A83G, CD2A84N & M, & CD50110 [5 tests]
These tests assume that dependent tasks will terminate while the main program executes a loop that simply

tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86 & 96, 86 & 96, and 58, resp.).

CD2B15C & CD7205C
These tests expect that a 'STORAGE_SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.

CD2D11B This test gives a SMALL representation clause for a derived fixed-point type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.

CD5007B This test wrongly expects an implicitly declared subprogram to be at the the address that is specified for an unrelated subprogram (line 303).

ED7004B, ED7005C & D, ED7006C & D [5 tests]
These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.

CD7105A This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK--particular instances of change may be less (line 29).

CD7203B, & CD7204B
These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD7205D This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.

CE2107I This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid. (line 90)

CE3111C This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.

CE3301A This test contains several calls to END_OF_LINE & END_OF_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD_INPUT (lines 103, 107, 118, 132, & 136).

CE3411B This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to a lack of available disk space, and the test would thus encumber validation testing.